The Paradigm Change from Thin to Rich Clients in Modern Web Applications

An analysis of how the MVC pattern is being shifted to the client side

for the

Bachelor of Science

in Applied Computer Science

at the Baden-Wuerttemberg Cooperative State University Stuttgart

by

Tim von Oldenburg

September 2012

Time of Project	12 Weeks
Student ID, Course	0834311, TIT09AIA
Company	IBM Deutschland MBS GmbH, Herrenberg
Supervisor in the Company	Daniel Suski
Reviewer	Paul Hubert Vossen

Research Question: *How can an IDE support the understanding and exploration of scope & context in a programming language to deepen understanding and help prevent errors?*

- IDE giving semantic support in code creation
- Understanding asynchronous control flow in IDEs // nope!!!
- Understanding and exploring scope and context

Contents

1 Introduction

Lorem Ipsum.

2 Research, Framework

2.1 Breadth and Depth of IDEs nowadays

Software development environments have been predecessed by general text editors, all starting with the „Mother of All Demos“ (cite: moggridge).

Alan Kay was also involved in the Smalltalk programming language, which was the first one to be accompanied with a class browser.

Today, IDEs make use of many general UI patterns and adapt them to their needs. The tree view, for example, is used in class browsers as well as „project explorers“, which serve as a hybrid between file browsers and project-related, non-file nodes (logical nodes) within a software project.

The IDE landscape is today more differentiated than ever, ranging from minimal, purpose-specific environments like Processing to huge, general-purpose, commercial environments like Visual Studio. Those different IDEs serve the needs of different developers and development situations. But still, it seems like there are many niches that are yet to be filled with new IDEs. Especially the area of web development (frontend development) is seeing many newcomers, for example Github's Atom Editor, Adobe's Brackets and Eclipse Orion, all based on Node.js and other web technologies.

- A Short History of Code Creation & IDEs; Alan Kay, smalltalk, etc

2.2 Design Patterns in IDEs

- Graphical User Interface, general patterns: navigation, integration of different tools. pattern elements: source code, the gutter, highlighting
- Mouse and keyboard (shortcuts) as input

- Modes (vim, larry tesler against modes, diff. configurations in eclipse, on-the-fly hide/show in sublime text/atom etc)

2.3 Programming concepts that are important

- Runtime & Creation Time
- Syntax & Semantics
- Scope & Context

2.4 Survey & Interviews

- Existing patterns that are relevant to mine
- Goals: understanding, less errors/mistakes

3 Design & Process

3.1 Process

- Ideation & Concept
- Prototyping: esprima, atom, brackets, devtools?
- User Testing, Probe (1 week)
- Interview, evaluation in quantity and quality

3.2 Existing solutions & inspiration, similar solutions (IDEATION)

- Syntax Highlighting (similar)
- Indentation (similar)
- Crockford's context coloring (existing, inspiration)
- Theseus' grey colouring for code that hasn't been called

3.3 IDEATION

To support the ideation phase, the author created a collection of existing UI components within IDEs. Those components were written down on post-it notes.

The components were used a seeds for *seeded brainstorming*: for each of the components, the author tried to imagine solutions that are similar, related to or based on the respective component.

Most of the ideas that came out of the brainstorming session made use of multiple components, for example the *context path* which is described further down: it made use of a status bar as well as the code editor.

Most of the ideas of the brainstorming phase made it into first sketches. The sketching happened with two different approaches, depending on if the code editor was involved or not.

For ideas that involved the code editor, it was important that the author could work with real, functioning code. Therefore, two sample JavaScript applications were created to work with:

- A small web server application, that would parse a markdown-formatted text file and render it into an HTML template. The application would run on Node.js and represents a typical control flow for e.g. a blogging engine (content + template = site).
- A client-side script (runs in a browser) that fetches JSON data and presents them on a website, by the click of a button. This script represents typical client-side UI code, connecting a button event to a function and presenting the result in the UI.

Both applications were written in different styles: the server-side application decouples the different tasks by putting them into different functions (as far as it makes sense), whereas the client-side application nests all function definitions inside each other, resulting in nearly one function definition in each line, and deeper indentation (ergo: higher code complexity).

A good solution for this design problem should address both cases.

Printouts of the two JavaScript files served as a basis for ideation *within source code*.

For concepts that would mainly work with other UI components, such as a sidebar, or such concepts that would introduce new UI components, blank paper was used for sketching.

3.4 CONCEPTS

- *Context Path* - a path view of the context tree, similar to that of a selector path in an HTML editor (screenshot!). The context at the position of the cursor would be shown in a status bar. By hovering over a context level, the corresponding source code would be highlighted in the source editor.
- *Context Graph* - similar to a class browser, the context graph would represent a tree view of the application's context(s). This could be implemented as a sidebar or panel.

- *Context Colouring* - similarly suggested by Crockford, the source code can be coloured in depending on its context (level). Crockfords variation is meant to replace syntax highlighting; one could, as well, complement syntax highlighting by colouring in the background (as e.g. Theseus does). 50 Shades of Grey.
- *Inspect Context* - comparable to DevTools' *Inspect Element* function, the user can right-click into the source code and choose *Inspect Context*, which opens a panel that shows global variables, current local variables as well as the value of this.
- *Gutter Context* - any change of context or scope is indicated in the code editor's gutter (similar to JSHint).
- *Quick Inspect* - similar to Brackets' *Quick Edit* feature, the value of this could be inspected inline.

3.5 PROTOTYPING

3.6 USER TESTING, (Probe) (1 week)

3.7 INTERVIEW, evaluation in quantity and quality

4 Conclusion, Reflection

- Crockford's „context colouring" https://www.youtube.com/watch?v=b0EF0VTs9Dc&noredirect=1

Bibliography

List of Figures

List of Tables

List of Listings

Appendix